National Research Council Canada
Conseil national de recherches Canada

Division of Electrical Engineering
Division de génie électrique

ERB-988

Integrity Constraints of Manufacturing Databases and their Processing

A. Famili and F.B. Vernadat

February 1986

Canada

NRCC No. 25660

86-06458/

Copyright 1986 by
National Research Council of Canada

Permission is granted to quote short excerpts and to reproduce figures and tables from this report, provided that the source of such material is fully acknowledged.

Copyright 1986 par
Conseil national de recherches du Canada

Il est permis de citer de courts extraits et de reproduire des figures ou tableaux du présent rapport, à condition d'en identifier clairement la source.

Additional copies are available free of charge from:

Editorial Office, Room 301
Division of Electrical Engineering
National Research Council of Canada
Ottawa, Ontario, Canada
K1A 0R8

Des exemplaires supplémentaires gratuits peuvent être obtenus à l'adresse suivante:

Bureau des publications, Pièce 301
Division de génie électrique
Conseil national de recherches du Canada
Ottawa (Ontario) Canada
K1A 0R8

INTEGRITY CONSTRAINTS OF MANUFACTURING DATABASES AND THEIR PROCESSING

A. Famili and F.B. Vernadat

ERB-988

NRCC No. 25660

February 1986

TABLE OF CONTENTS

		Page
ABSTRACT		v
RÉSUMÉ		vi
1.0	INTRODUCTION	1
2.0	TAXONOMY OF INTEGRITY CONSTRAINTS	4
	2.1 Integrity Constraints in the Real World	4
	2.2 Manufacturing Integrity Constraints	10
	2.3 Integrity Constraints in the Database Environment	11
	2.3.1 Integrity Constraints and the Relational Data Model	11
	2.3.2 Static and Dynamic Integrity Constraints	13
	2.3.3 Global and Local Integrity Constraints	13
	2.3.4 Compiled vs. Interpreted Integrity Constraints	14
3.0	REVIEW OF PREVIOUS WORK IN INTEGRITY CONSTRAINTS	15
	3.1 Early Work	15
	3.2 Other Approaches	18
	3.3 More Recent Work	18
4.0	INTEGRITY CONTROL IN RELATIONAL DBMS'S	21
	4.1 Expression of Integrity Constraints	22
	4.1.1 Data Typing	22
	4.1.2 Referential Integrity	27
	4.1.3 Semantic Integrity	31
	4.2 Checking of Integrity Constraints	33
	4.2.1 Immediate Checking	33
	4.2.2 Selective Checking	34
	4.2.3 Delayed Checking	34
	4.3 Enforcement of Integrity Constraints	35
	4.3.1 Accept/Reject Enforcement	35
	4.3.2 Enforcement with ELSE Clause	36
	4.4 Management of Integrity Constraints	36
	4.4.1 Integrity Constraints Maintenance	37
	4.4.2 Integrity Constraints Enabling and Disabling	40
	4.4.3 Integrity Transactions and Validation of Integrity Checkpoints	41
5.0	RECOMMENDATIONS FOR DBS/R	45
	5.1 Current Status of DBS/R	45
	5.2 Integrity Constraint Expression Reconsidered	46
	5.3 Proposal for Extension of DDL in DBS/R	47
	5.4 Proposal for New DML Primitives	49
	5.5 Other Integrity Management Capabilities	50
6.0	SUMMARY AND CONCLUSION	51
	REFERENCES	53

TABLE OF CONTENTS (Cont'd)

	Page
Additional Reading List	55
APPENDIX A	57

FIGURES

1	Hierarchical structure of the integrity constraints	5
2	Specific levels of integrity constraints	6

ABSTRACT

Today's manufacturing cost reduction and competitiveness largely depends on the application of automated manufacturing. This automated manufacturing could be even more efficient if computerization of the process is taken one step further - to integration of the design and manufacturing processes by means of or through a well-structured database.

This is an extensive report in which several important issues related to integrity constraints in relational databases are examined. The literature review provides a summary of what has been done in this area by other researchers. In addition, some of the concepts proposed, developed and even implemented before, are introduced. An indepth look at the classification of integrity constraints is performed by examining the real world, engineering and manufacturing worlds as well as database world. This classification is supported by various manufacturing examples for each type of integrity constraints. We further discuss the issues of how to express and manage integrity constraints. Different features of three commercial types of databases, namely INGRES, SQL and RIM, as well as DBS/R (a prototype system developed at NRC) are discussed. This will lead us to the development of a conceptual framework for implementation of integrity constraints in DBS/R.

RÉSUMÉ

De nos jours, la fabrication automatisée est un facteur clé pour la réduction des coûts et l'augmentation de la compétitivité de bien des entreprises. La fabrication automatisée peut être encore plus efficace si on pousse un peu plus loin l'informatisation, c'est-à-dire jusqu'à l'unification de la conception et de la fabrication au moyen de bases de données bien structurées.

Ce document est un rapport détaillé dans lequel on examine plusieurs questions importantes portant sur les contraintes d'intégrité dans les bases de données relationnelles. Un survol de la documentation publiée résume ce qui a été fait dans le domaine par d'autres chercheurs. Sont aussi présentées certaines notions proposées, mises au point et même mises en oeuvre auparavant. L'étude approfondie de la classification des contraintes d'intégrité se fait par le biais d'un examen des caractéristiques du monde réel, du milieu du génie et de la fabrication ainsi que des bases de données. Cette classification s'appuie sur divers exemples tirés du milieu de la fabrication pour chaque type de contrainte d'intégrité. On se penche enfin sur les façons d'exprimer et de gérer les contraintes d'intégrité. Les diverses caractéristiques de trois types de bases de données commerciales, soit INGRES, SQL et RIM, ainsi que de DBS/R (prototype mis au point au CNRC) sont étudiées. Les résultats conduisent à l'élaboration d'un cadre conceptuel pour l'implantation des contraintes d'intégrité dans le système DBS/R.

INTEGRITY CONSTRAINTS OF MANUFACTURING DATABASES AND THEIR PROCESSING

A. Famili and F.B. Vernadat

1.0 INTRODUCTION

A database is an accumulation of interrelated data items that is used for different purposes in an enterprise to carry out numerous but limited functions. The database may include information objects or entities and relationships that exist between these entities. The entities and relationships, in a database, are presented in some physical way - by records and pointers or other methods, depending on the database management system.

A database management system (DBMS) provides the required access to the database for the user or the application program, and also provides many facilities such as data retrieval and performs the requested operations on the stored database. The most important characteristic of a database system is that it supports a number of different applications, which can be on-line, batch, or both.

Today's manufacturing cost reduction and competitiveness largely depend on the application of automated manufacturing. This automated manufacturing could be even more efficient if computerization of the process is taken one step further - to integration of the design and manufacturing process by means of or through a well-structured database.

Computer integrated manufacturing depends heavily on data communication, which starts with the preliminary design of a product and continues through production operation, especially in a flexible manufacturing environment, and it ends at the last stage of production, which involves assembly of or packaging a system. Instead of recreating and reinterpreting the data at each stage of the production process, the user enterprise can avoid duplication by organizing and storing in a computer such information as geometry and dimension of parts, part number, parts list, manufacturing resources, assembly procedures, scheduling, and inspection results. This convenience is not obtained by anything but an efficient database.

Since the majority of manufacturing databases are to serve multiple applications, where designers, managers, system operators, etc., will have access to a common datapool, precautionary routines must be provided to

monitor the integrity of the data as they are entered. In other words, multiplicity of users with a different range of integrity requirements, which in general is attributed to the characteristics of the database, has been forcing researchers to find a suitable approach.

Here, the term integrity is used in database contexts with the meaning of accuracy, correctness, or validity of the data. The problem of integrity is the problem of ensuring that the data in the database are accurate, that is, the problem of guarding the database against invalid updates. Invalid updates may be caused by errors in data entry or by mistakes on the part of the operator or the application programmer.

An example here will illustrate the problem: let us assume a mid-size company which fabricates pressure tank vessels is using an existing manufacturing database. The engineering group of this company, which consists of 15 employees, will be using this database. Since the company is primarily involved in fabricating pressure tank vessels, there are certain design limitations such as thickness of the metal sheets, thickness of the welding seam, end curvature of the tank, location of the safety valves, etc., that must be met during the design stage and are usually determined by certain standards and should be automatically checked during the design stage of the product. As a result, a section may be added to the database system so that the integrity of certain values could be checked while different staff members are using a common database. For example, if the distance between the center of the tank and the safety valve cannot exceed a certain value, each time that a designer uses this database, the value that is entered or updated must be checked against the limits.

However, there may be some special cases in any application when none of the integrity constraints have to be activated. In other words, the designer has free hands on the design specifications and he does not have to worry about invoking any constraints. This is the case when integrity constraints exist in a database; the DBMS should have the capability to permit the users to by-pass some rules at the execution time.

The study discussed in this paper is concerned with the expression, validation, maintenance, and management of integrity constraints in a manufacturing database and especially for engineering relational database environments.*

---

*Through this report, detailed knowledge of database principles, and more specifically of relational database theory, is assumed on the part of the reader.

As discussed earlier, the need for ensuring validity and consistency of data stored in the database is of great importance in such areas as design and manufacturing for which many data items are severely constrained. Therefore, in addition to being a data definition, storage, and retrieval system, a manufacturing database management system (DBMS) must support the following functions:

- expression of integrity constraints: The data definition language (DDL) of a DBMS must be augmented to support the declaration of the integrity rules or assertions on data items (at least for those that can be expressed in a formal language).

- checking integrity constraints: Data insertions, deletions, or updates may alter the database integrity. Therefore, rules in which they are involved must be checked for validity.

- maintenance of integrity constraints: If the integrity rule has been evaluated to be true, the data updates are committed in the database. If the rule is evaluated to be false, the current updates are rejected or undone.

- enforcement of integrity constraints: Sometimes integrity checks must be held at some given times and not during other times. Also, for some performance reasons, it is desirable to check a set of rules at some specific instants rather than for each update (concept of integrity transactions). Thus, the data manipulation language (DML) of the DBMS must be augmented in order to enable or disable specified integrity rules. In some systems, the DML may allow insertion of new rules and modification or deletion of existing rules.

In this report, we discuss many details of DBMS features for integrity management in the specific framework of design and manufacturing databases, and we make recommendations for implementing them in DBS/R, a relational database environment for manufacturing databases developed at the National Research Council of Canada.

2.0 TAXONOMY OF INTEGRITY CONSTRAINTS

Depending on how one considers the integrity constraints, several classifications can be made. In this section, we successively consider integrity constraints in the real world, in the engineering and manufacturing worlds, and finally in the database world.

2.1 Integrity Constraints in the Real World

At a given time, a database is in a consistent state if the values of the stored data are in accordance with some realistic situation of the real world that the database is supposed to model. In other words, a database is in a consistent state if the values of its data items are plausible and consistent with each other for a given universe of discourse.

Therefore, the integrity problem is a subjective one, since it takes into account the way that people understand a real world situation and its modeling in the database. For example, 360° is a valid number for the base axis rotation of any articulated robot, while it may not be valid for some intermediate joints of a robot arm.

In fact, the integrity problem has a hierarchical structure (Fig. 1) with several levels of specific classification (Fig. 2). The deeper one goes into the structure, the more specific integrity rules one would require.

At each level of the structure, an integrity rule, which is in fact a predicate, has to be verified by a subset of the database elements so that the whole database remains consistent. Usually, the predicate is a numerical or a Boolean expression which restricts the set of possible values of some data items when it is evaluated to be true. We illustrate this point hereafter.

As we know, in database terminology, elements of the real world can be modeled as:

- **entities** which represent objects of the real world having existence of their own and referred to by a name
- **relationships** which are associations between several entities
- and finally **attributes** which are properties of entities or relationships

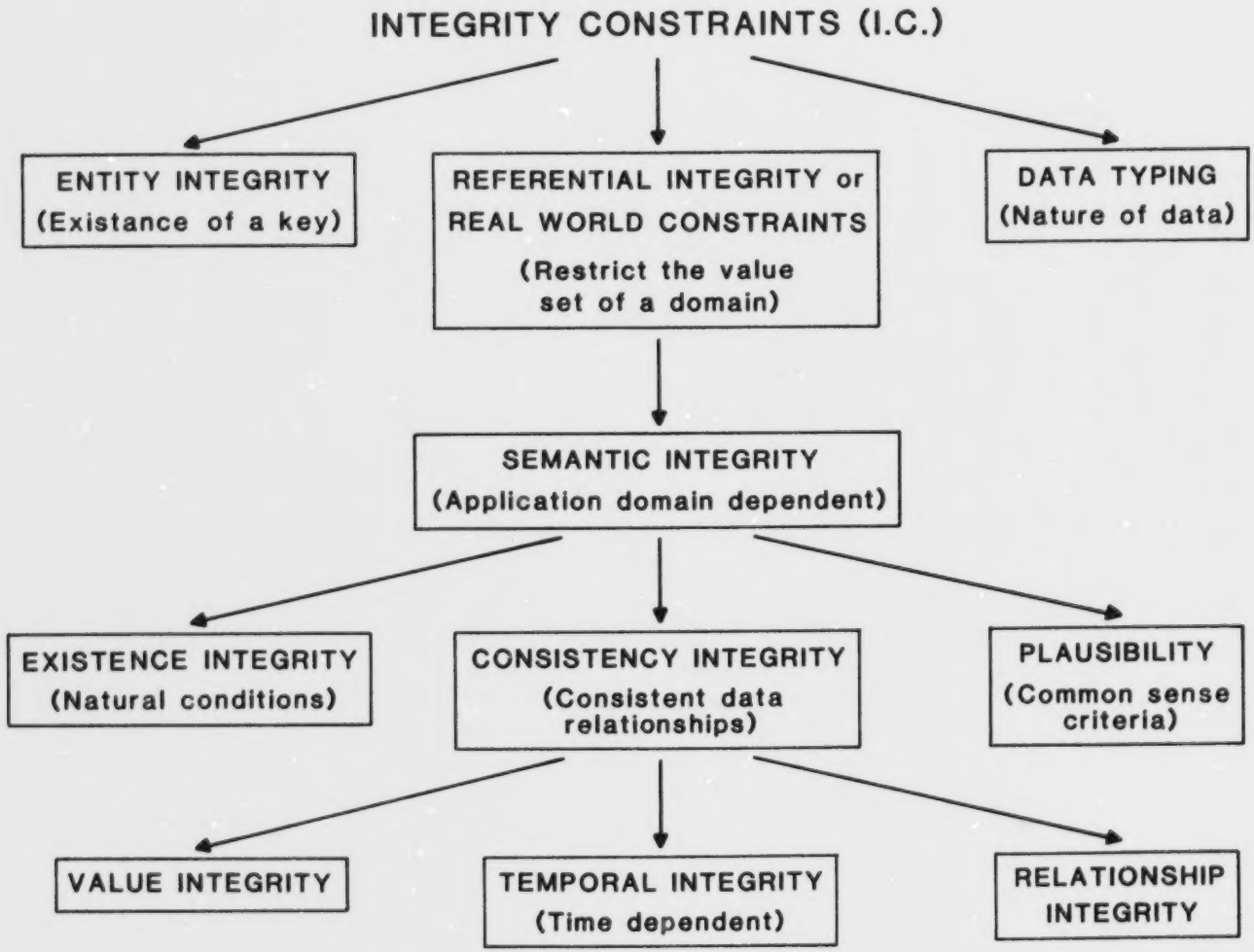

Figure 1. Hierarchical structure of the integrity constraints

Application dependent	Application level	Functionality of data Physical laws Optimization criteria
	Semantic level	Semantics of data Meaning Value Relationship Time Dependency
Application independent	Referential level	Domain of data Interval Range Subrange list of possible values subset
	Data typing level	Nature of data (simple, structured, abstract type) integer real character string record array text sound image bit

Figure 2 Specific levels of integrity constraints

Looking at Fig. 1, we can see that in any case, manufacturing or otherwise, there are three types of integrity constraints:

- entity integrity
- data typing
- referential integrity

Entity integrity has been defined by Date (1). It ensures that each entity of the real world to be represented in a database can be referred to by a unique and non-null identification. Usually, the identifier is called the key. This is one attribute or group of attributes of the entity, the value of which designates one and only one occurrence of the entity type. In some cases, several keys can be defined for the same entity type (candidate key).

Example: Let a CNC machine be an entity name, and MMN (machine model number), number of axes, turret capacity be its attributes. We write:

CNC Machine (MMN, No. of axes, turret capacity)

Obviously, two CNC machines may have the same number of axes or the same turret capacity, but usually a machine model number, which is a combination of characters and digits, is used by a given manufacturer and is unique. For instance, VDF400C is a CNC machine build by Derliken Boehringer.

In this example, we assume MMN to be the key. This must be declared at the DBMS.

Data typing is another general form of integrity constraints. According to Fig. 2, it is the least specific level of integrity since it is at the bottom level. Each attribute has a data type associated with it that defines the nature of its possible data values. There are many types of data in the real world that could be classified as simple data types (integers, real numbers, characters, Booleans), structured data types (strings, records, arrays), and abstract data types (text, sound, image, series of bits).

Example: Consider the previous example again. CNC machines are referred to by the number of axes as a positive integer (e.g., 3), by a model number as a character string (e.g., VDF400C, etc).

Referential integrity or real world integrity (also called domain integrity) is restricted to the possible set of values (also called domain) of an attribute. It takes into account part of the semantics of data (usually general, universally accepted semantics about the definition of data). This corresponds to the referential level of Fig. 2. Integrity rules of this class are used to exactly define the set of all possible values of some attributes in terms of a restriction of their data types. In other words, referential integrity is really part of the general definition of the attribute.

Examples: Distance is always a positive number, the value of a day is in the range of 1 to 31, etc.

The classification up to this point is very general and it is only related to the global nature of data (their type and their set of possible values). To be more specific, we have to move to the semantic levels of Figs. 1 and 2, and we can define more precise rules related to the real meaning of data for a given universe of discourse. Those rules are called semantic integrity constraints and obviously they are application domain dependent since the meaning of data can vary from one field of application to another.

From now on, we assume that we are considering a given universe of discourse.

There are three types of **semantic integrities** (Fig. 1):

- existence integrity
- plausibility integrity
- consistency integrity

Existence integrity is used to define the real world, natural situations, or facts in a given application domain. It relates to the very nature of an entity as understood by specialists in the field. Usually it occurs in the definition of the list of relevant attributes for a given entity.

Example: In geometric modeling, if we want to store information about a line, we have to indicate that it has exactly two ends which are two points. A point is defined by its three coordinates (x, y, z).

Point (Pointid,x,y,z)
Line (Lineid,startpoint,endpoint)

Plausibility integrity ensures validity of a data value based on common sense with respect to the application domain considered. In fact, it restricts the set of values of a domain for a given attribute. Therefore, it is more restrictive than referential integrity since it adds semantics of the application field to the general semantics of the data.

Example: The working voltage in direct current of a ceramic disc capacitor may be in the range of 500-6000 V while for a ceramic plate capacitor it is in the range of 10-100 V.

Consistency integrity is used to verify constraints involving relationships between two or more attributes of the same or different entities. They describe situations of a universe of discourse for which the values of some quantities are dependent on the values of others. Consistency integrity constraints are part of the semantic level of Fig. 2 and they have three types (Fig. 1):

- value integrity
- relationship integrity
- temporal integrity

Value integrity is used to impose a specific value to a specified attribute of a given entity or relationship in accordance with the value previously given to another attribute of the same or another entity or relationship.

Example: In design engineering, to indicate that two cylindrical pipes P1 and P2 of a building have the same diameter; Diam., we write:

Diam(P1) = Diam(P2)

Relationship integrity describes the situation for which the value

of a specified attribute of a given entity of relationship is functionally dependent on the values of other attributes of different entities or relationships. In other words, the value of an attribute is computed from the value of other attributes.

Examples:

(1) In mechanical engineering, if a mechanism consists of many subsystems, total mass of the mechanism is the sum of the masses of the elements;

$$M = \sum_i m_i, \qquad M = \text{Mass}, \quad m_i = \text{mass of element } i$$

(2) Consider the following two entities:

Point (Pointid, x, y, z)
Line (Lineid, Startpoint, Endpoint)

A relationship integrity rule is: "Verify that each endpoint of each line exists in the list of points."

Temporal integrity takes into account the time dependency of some constraints on data.

Example: The value representing the inside diameter of a cylinder after being bored for the ith time must be greater than the previous diameter.

Finally, Fig. 2 shows a fourth level or application level which is concerned with the functionality of data, i.e., the real specificity of the problems under consideration. Here, since it is in the scope of the application, it is not discussed in this section, which remains very general, but it is different from the next one, especially concerned with design and manufacturing.

The classification of integrity constraints just discussed can be applied to any application environment. Therefore, there is the need to implement mechanisms in DBMS in order to be able to process such constraints.

Remark: It is important to notice that consistency integrity rules such as value integrity and relationship integrity constraints introduce redundancies in the database when the attributes involved are related by constraints of the form:

$a_j = f(a_i)$ $\quad a_i, a_j$ attributes

or $a_j \leq g(ai)$ $\quad 1 \leq i \leq n, j\ [1,n],$ n: number of attributes

Unfortunately, redundancy is in contradiction with the theory of normalization in relational databases (2, 3). Therefore, it must be well understood that, in order to implement semantic integrity checking in a database, unavoidable redundancies are introduced, but they are controlled as long as the associated integrity rules exist in the system.

2.2 Manufacturing Integrity Constraints

Since most of the examples of the previous section were obtained from the engineering and manufacturing fields, our emphasis for integrity constraints will be related to these fields. In fact, engineering is one of the most important fields in terms of integrity constraints management since design and manufacturing involve many (complex) rules and criteria for feasibility, performance, optimization, quality control, productivity, efficiency, cost reduction, etc.

Moreover, talking about the integrity problem, engineering has its own area of specialization, which is represented by the application level of Fig. 2. At this level, we distinguish four types of engineering constraints (although other authors may propose a different classification):

- functional
- design and manufacturing constraints
- feasibility constraints
- performance constraints

Functional constraints can be defined as governing laws for the application domain. They are usually physical laws or mathematical expressions which can invoke high-level complex sequences of computations. For example, analog circuit analysis involves many successive uses of Ohm's and Kirchhoff's laws. Usually, due to their complexity or the logic associated with their application, those functional constraints are included in an application program of database. In this case, the DBMS assumes that the data it receives from the external program are correct and consistent.

Design and manufacturing constraints represent the know-how of the firm or of its personnel (designers, machinists, engineers, etc.). Examples of such constraints can be based on well-established knowledge from machine manuals (e.g., a feed rate between 0.005 and 0.400 ipr for a turning machine tool for machining carbon sheet), standards and codes (International standards ISO, IEEE, etc., building codes, etc.) or they can be

based on experience or personal judgement (artist's view, designer's style, ...). Example of designer's style rule may be that for designing a table, the length should not be greater than twice its width.

Feasibility constraints are used to insure that the product is realizable. They may invoke criteria reflecting current technological limitations (tolerances in fabrication, quality control, etc.), resource limitations (labor rate, time estimates, function costs, material waste, ...) or artifact consistency constraints such as spacial conflicts (two objects interfering in space), conservation laws (for current mass, energy, fluid, ...), and topological connectivity (two connected pipes must have same diameter at their connecting ends).

Performance constraints in engineering are required for improving the product manufacturing, the product quality, and the product efficiency. Usually, the performances of a product are determined by analysis programs involving computations of one or several (complex) optimization criteria, computer simulation sometimes involving graphical representations, or by engineers based on their experience or their intuition.

At this level of classification (application level), we are really at the frontier between the DBMS and the application program responsibilities for integrity checks. Therefore, it is sometimes difficult to decide if the constraints should be incorporated into the DBMS or in application programs. However, we can state that if the integrity check is too specific or if it involves (excessive) procedural computations, it must be included in an application program. As database users, we must take into account that as more integrity rules are placed into the database, the system will become slower.

2.3 Integrity Constraints in the Database Environment

From the database point of view, a different classification of integrity constraints can be made which depends on the organization of the data structure. In this section, we only consider the relational data organization.

2.3.1 Integrity Constraints and the Relational Data Model

A relational database is a collection of relations identified by their names (1).

A relation is a two-dimensional data table, the columns of which are called attributes and take their values from a domain, and the lines of which are called tuples. We note:

Relation name (attribute1name,attribute2name, ...)

Example: Point(Pointid,X,Y,Z)
Line(Lineid,Stpt,Ept,Length)

The underlined attributes are the keys of the relations.

Due to this structure, four types of integrity constraints can be defined:

- single attribute integrity constraints
- single tuple integrity constraints
- intra-relation integrity constraints
- inter-relation integrity constraints

Single attribute integrity constraints are rules used to restrict the domain of an attribute.

Example: Rule; The length of a line is always positive.

Length > 0

Single tuple integrity constraints may involve several attributes of one tuple of a relation.

Example: Rule; The Z coordinate of all points is twice the y coordinate plus the X coordinate:

Z := (2 * Y) + X

Intra-relation integrity constraints may involve attributes from several tuples of a relation.

Example: Rule; The length of a new line is restricted to less than or equal to the average length of the other lines.

Length < Avg (Line. Length)

Inter-relation integrity constraints are general constraints involving attributes for various tuples of various relations of the database.

Example: Rule; The length of line is the Euclidian distance.

Length := SQRT (SQR(Xe-Xs) + SQR(Ye-Ys) + SQR(Ze-Zs))

2.3.2 Static and Dynamic Integrity Constraints

All the constraints given in paragraph 1 are static integrity constraints.

A **static integrity constraint** is in fact a predicate on the current status of a database which is time independent.

A **dynamic integrity constraint** is a constraint which applies when the database goes from an old state to a new one. In other words, the tuples of the new state depend on the tuples of the old one. A classical example is a rule which specifies that the salary of an employee can never decrease. Therefore, to define such a rule, we need to know the salary before and after the modification.

2.3.3 Global and Local Integrity Constraints

Global integrity refers to the integrity of the whole database.

However, it has been discussed (4) that, especially in design engineering, the semantic integrity of a database cannot be reached before the end of the project. There are mainly two reasons:

- Complexity of the design process: Hundreds of rules can be defined in the system. Some of them may cause the checking of others (causing effect). Therefore, imposing global integrity in any time on the database would be very time-consuming and even ridiculous since the same checkings could be performed several times and the performance of the system would be simply unacceptable.

- Incompleteness of the design database: During the design stage of any system, the contents of the database grow. Thus, some pieces of data required by some rules may be undefined if global integrity is permanently imposed. In these cases, the system can reject new data or detect inconsistencies or, even worse, it can crash.

Therefore, in any kind of database, manufacturing or any other, global integrity should be preserved at some moments. However, for performance and practical reasons, there could be other cases for which global integrity could not be verified at any moment at all. These are cases where local integrity can be defined on a subset of the database.

Local integrity refers to a subset of data of the whole database which is constrained by a subset of rules of the integrity rule set.

To explain the concepts of global and local integrity, let us consider the design of a car engine. Let us assume that there is a global rule stating that the diameter of a piston cannot exceed the internal diameter of its cylinder. Obviously, even if the piston has been designed, the rule cannot be checked until the design of the engine block is completed. However, many other rules for integrity can be expressed and checked during the design stages of the piston.

The global integrity concept is enforced in the case of a multi-user environment of the database. Suppose the engine block is engineered by one designer and the piston is designed by someone else. Both designers interact independently with the database, but they need their own set of local (or personal) design rules. Global integrity of the complete engine will be verified only at the end of their design work.

2.3.4 Compiled vs. Interpreted Integrity Constraints

From a computer science point of view, there are two ways of implementing integrity constraints depending on the nature of the DBMS.

If the DBMS is compiled and has a static data description language, all the rules have to be expressed in the database description and are known at compile time. Therefore, the database schema is static in the sense that it cannot be changed without recompiling the entire database structure. This solution assumes that the syntax of the DDL is rich enough to describe each type of integrity constraint. However, its advantage is efficiency of the DBMS code which provides better responses at run time.

If the DBMS is a database interpreter with a data dictionary, the schema is dynamic. Thus, new rules can be added and previous rules can be modified or deleted. The obvious advantage is flexibility of the database system, but its drawback is its processing speed since many computations are involved for interpreting the rules. RIM (5) is a CAD/CAM DBMS example with interpreted integrity constraint implementation.

3.0 REVIEW OF PREVIOUS WORK IN INTEGRITY CONSTRAINTS

The problem of integrity constraints has been investigated by many researchers in different fields of engineering and computer science. Different approaches have been taken and numerous solutions have been suggested and tested. Since one of the objectives of this section is to study the work that has been done in this area, whether directly or indirectly related to our work, we discuss some of the approaches and solutions proposed by other researchers that could be used in our work.

3.1 Early Work

The following is a summary of some of the literature that has been reviewed:

Pioneer work in the area of semantic integrity was done at Carnegie-Mellon University by Eastman and Fenves and their students (Lafue, Holtz, Rasdorf, and others), as we will discuss later.

Lafue (6) was perhaps the first person who investigated the automatic maintenance of semantic integrity with respect to engineering databases and proposed three basic principles. According to Lafue, first, the maintenance of integrity is delayed until strictly necessary. Second, integrity violations are temporarily tolerated, and finally, integrity constraints can be included as procedures in the record definition and automatically activated by the system if the concept of abstract data types is used. These ideas have obviously certain consequences that should be considered. For example, the consequence of delayed maintenance of integrity is that the database must be able to tolerate violations of its integrity, at least temporarily.

Automatic maintenance of semantic integrity was also considered by Frost et al. (7) in which they utilized first-order predicate calculus to introduce rules of semantic assertions in a conceptual schema constraint language called SCHEMAL. SCHEMAL is a decidable subset of first-order predicate calculus which is based on the simplified binary relation (SBR) view. Obviously, the integrity of a database can be maintained by checking that all new instantiations which are implied by a new triple (a triple is a set of data representing a binary relationship and consists of three fields) would be true if the triple was in the database. If so, the triple may be inserted into the database, otherwise it should be rejected. Five stages of automatic maintenance of semantic integrity introduced in this research were tested for automatic maintenance of integrity of a binary-relational database and it was found that the SCHEMAL had some limitations and was not user friendly.

The problem of immediate checking vs. delayed checking set up by Lafue (8), in which he suggested that some variables of the constraint may be operated while others may not, is interesting to consider. Usually a constraint need not be checked immediately after operation on instances of its independent variables, if the corresponding instances of its dependent variables are of no interest to the current user/application. Instead, this checking can be delayed until the dependent instances become of interest, i.e., are accessed, since they are the only instances to be affected by this constraint. This approach, which is called delayed integrity checking, is appropriate only if integrity dependencies can be defined. It also assumes that (i) cycles of dependencies are localized in the database and (ii) the history of data values is not important.

Delayed integrity checking fits some manufacturing databases very well. It is obviously cheaper than immediate checking of database accesses.

However, this approach has not been accepted in developments of all CAD database systems. As an example, in ROSALIE, a LISP software tool to support a CAD semantic database developed by Cholvy et al. (9), different rules can be assigned to data objects (classes or relations). For each update, associated rules are verified as soon as possible, to prevent an operation if it leads to an incoherence. In other words, immediate checking was the approach taken for ROSALIE along with the concept of constraint propagation.

The studies done by Bernstein et al. (10), in which they proposed a class of assertions that can be efficiently enforced, introduce suitable tactics and then analyze the compile time and run-time enforcement algorithms. Although this approach is time-consuming and complicated, it probably offers a hope in developing a semantic integrity subsystem with acceptable performance.

The confidence that the contents of a manufacturing database represent a possible real world situation may be hard to measure. Several factors can affect this confidence. One important factor is the aspect of semantic integrity constraints, that is, constraints which restrict data values so that the set of values of a database is semantically consistent with the reality being modeled.

As discussed by Eastman et al. (4), two major types of integrity constraints can be conceptually distinguished. The first type specifies the cardinalities of data records, that is, the number of elements or data fields which can exist in the records. An example of this type, which is called **existence integrity constraints**, is a joint of a robot that can

have either rotary of prismatic degree of freedom. The second type of constraints states the relationships between the values of the related data. For example, rotation of the base axis of most articulated robots cannot exceed 360°. These are called **value integrity constraints.**

The subject of global and local integrity also introduced by Eastman et al. (4) considers that the integrity of the entire database does not necessarily have to be guaranteed before and after every transaction. As a matter of fact, this complete guarantee between the transactions is generally impractical in design and manufacturing databases for several reasons. Usually the semantic integrity of a database for a design project will not be complete until the end of the project.

As discussed by Eastman (4), the lack of global integrity throughout most of design forces the introduction of the concept of local integrity in terms of individual variables and individual constraints. Local integrity directly translates into integrity transactions. A transaction is the execution of a collection of operations on a set of variables, during which the integrity of these variables with respect to a set of constraints is not guaranteed, but before and after which such integrity is either true or not applicable. Locality of integrity can add flexibility to the scheduling of checking with respect to the maintenance and vice versa. Different consequences of a given update can be checked at different times after the update.

Integrity constraints in solid shape modeling and whether useful characteristics of constraint management should be managed during data entry or during data interpretation were discussed in another reference by Eastman and Preiss (11). They believed that well-formedness checking at entry was preferable for interactive systems.

Other forms of expressing integrity constraints is the logical form, as discussed by Nicolas (12), or classified as integrity assertions, discussed by Weber et al. (13). These are basically classified as

- state, prestate, and transition assertions
- operation dependent assertions
- immediate, delayed, and user-invokeable assertions
- strong, soft, and self-correcting assertions
- monotuple and multituple assertions

The above assertions as well as static integrity testing could be simultaneously or separately used or added during development of a database system.

3.2 Other Approaches

The approach taken by Cremers et al. (14) in which an integrity monitor called AIM was developed to introduce important maintenance strategies is to extend the Ingres user interface with a non-trivial Semantic Control Mechanism. All constraints, in this approach, were assumed to be immediate in the sense that they were supported to hold after every elementary update. The update request, in this approach, will either not take effect or all changes implied by a violating request are undone.

This approach is a refinement of that taken by Stonebraker (15). In the embedded approach of Stonebraker, defined integrity constraints are stored in a catalogue, in proximity to the description of the data objects to which they refer. Constraints are implemented by means of query modification and automatic integration of the constraint in the request guarantees the prevention of certain integrity violations.

3.3 More Recent Work

Much recent work that was published in 1984 and 1985 is mainly concentrated around integrity constraints in engineering design databases. These databases are considered to be the main integration element between the various design processes in a manufacturing environment.

The Semantic Integrity (SI) or transaction validation model introduced by Badal (16) consists of two phases: reading SI assertion argument values (SIREAD) which are not generated by the executing transaction and evaluation of SI assertions (SIEVAL). Three investigation techniques were proposed and tested in this work, as follows:

- pre-execution time SI enforcement, in which the transaction is first validated and only then executed.

- run-time SI enforcement in which the transaction execution and its validation are interleaved.

- pre-execution-run time method in which the transaction is also interleaved with its execution. Among the three above-mentioned techniques, the pre-execution run time SI enforcement method showed significantly lower cost and more potential for applications.

The range of integrity constraints that can be specified on a schema has been expressed in a network DBMS by Parkash et al. (17). The integrity constraints which were classified into three types of value sets, record type, and set type were implemented by a proposed mechanism into the

DBMS. This approach requires explicit statement of all integrity constraints in the subschema. As a result, whenever a transaction is performed, all the integrity constraints of the schema applicable to that transaction (whether it is update, insert, or anything else) should be carried over so that it will never allow an operation which violates integrity constraints to succeed.

In another recent work, Ling et al. (18) introduced Insertion, Indifference, Deletion, and Modification rules that could be implemented in conjunction with other methods to achieve overall improvement in performance. The idea behind introducing these rules was simplification of integrity assertions on the database in order to make the constraints easier to verify. They concluded that an integrity subsystem will be more efficient if certain constraints that are not affected by a specific change in the database state can be excluded automatically from the integrity check.

One way to manage integrity constraints is perhaps to classify the constraints such as the classification made by Bertino and Apuzzo (19) in which integrity constraints were listed as Tuple Constraints, Relation Constraints, and Multi-Relation Constraints. Based on this classification, three different ways to check integrity constraints, which depend on the type of assertion and on the structure of the transaction, were proposed. If the transaction is enforced after each tuple-update (or update request) and the enforcement action is specified to an abort action, the system will not perform the tuple-update (or update request). On the other hand, if enforcement is delayed to the end of the transaction, the system must allow all updates to be performed and then the correct ones are undone. In addition, if a delayed constraint is found to be invalid the system must be able to identify the update-request or tuple-update that caused the violation.

The work of Simon and Valuriez (20) is another example of classification of integrity assertions. Here, integrity assertions were grouped into individual assertions which are mono-relation contraints, set-oriented assertions that are multi-variable, multi-relation constraints, and constraints involving aggregates. The enforcement algorithm for the above constraints could be executed on an update request. Although this approach looks preventive, it exploits assertion simplification strategies for each class of the above-mentioned assertions.

Database Integrity Block (DIB) is another idea proposed by Lilien and Bhargava (21) in which a scheme was proposed for maintenance of the database integrity. This scheme is based on the concept of integrity assertions (AI), a rollback and recovery process, a differential file, and a

recovery block. In this approach, after a database is updated by transactions, its integrity must be verified by evaluation of a batch of semantic integrity assertions. This evaluation of integrity assertions includes a number of database pages to be transferred from the secondary storage to the fast memory where the verification will be done.

Finally, direct enforcement of integrity constraints, a work done by Fenves and Rasdorf (22), is another solution to the problem of integrity constraints. This approach, which encourages direct incorporation of integrity rules, is basically a direct extension of the single-attribute integrity found in most current relational DBMS's. However, in current relational DBMS's, the integrity rules are defined on the schema and are active as a relation is instantiated.

This approach was pursued and published in two reports by Fenves and Rasdorf (23) and Fenves (24). As suggested in this approach, constraints are directly incorporated into the integrity rules enforced by the database, so that transactions (insertion and updates) on tuples satisfying the constraints are accepted, while transactions violating them are rejected. This is accomplished by Pascal procedures attached to relations by means of status attributes containing Boolean values. This mechanism is a direct extension of the single-attribute integrity rules found in most relational database management systems and is essentially the approach taken in enforcing semantic integrity. However, this mechanism lacks some of the required flexibilities. First, in early phases of design, when information is gathered from many sources and tentative solutions are obtained, it is not realistic (or even feasible) to require that all constraints pertaining to the completed design be immediately satisfied on the first trial design, nor is it realistic to assume that all of the data needed to evaluate the constraints will be available. According to Fenves' (24) proposal, three new commands are added to DBMS commands. These are Invoke, Activate, and Deactivate. The Invoke command brings the status attributes up to date from all previous transactions, while the Activate command insures constraint compliance for all subsequent transactions. Finally, Deactivate suspends the enforcement process. These features are discussed in more details in Section 4.4.

4.0 INTEGRITY CONTROL IN RELATIONAL DBMS'S

Integrity control in most relational DBMS's can be performed by way of checking, as indicated earlier. However, the expressions that are used for integrity constraints are the most important elements of asserting the problem. In other words, well-managed DBMS design and well-organized assertion techniques can considerably improve system reliability.

Integrity control in relational databases involves four major functions on the part of the DBMS as mentioned in the Introduction:

- expression of integrity constraints
- checking integrity constraints
- enforcement of integrity constraints
- management of integrity constraints

The aim of this section is to discuss successively these four issues in detail. For the purpose of clarity, we will illustrate the discussion by considering four different implementations of relational DBMS's which have been developed as part of research projects. The four DBMS's are

- Ingres
- SQL (formerly system R)
- RIM
- DBS/R

Ingres is a multi-user database management system developed primarily for teaching and research at the University of California at Berkeley. Later it was extensively improved and is currently a commercial product available from Relational Technology Inc. for a wide variety of applications. Ingres can support application programs written in various high-level languages (C, Pascal, COBOL, FORTRAN). Users can also interact with Ingres using QUEL, an interactive non-procedural query lanaguage.

SQL (Structured English Query Language) is the integrated data language of System R, an experimental relational DBMS developed by IBM Research Laboratory, San Jose, CA (25-27). SQL is usually considered as a reference by researchers on relational data language and many extensions or improvements have been proposed. Commercial systems based on SQL include ORACLE from Relational Software Inc. and SQL/DS from IBM.

RIM (Relational Information Manager) is a DBMS that was developed under the IPAD project (Boeing Computer Services Co. (5)). This is one of the first DBMS's developed for manufacturing and engineering activities. RIM incorporates some advanced features for integrity constraints

management. Although the data language of RIM is a SQL-like language, it offers both algebra-level and calculus-level data manipulation commands through an interactive interface. RIM can be concurrently accessed by several users/application program for retrieval but allows one user for update.

DBS/R is a database system developed at the National Research Council of Canada for manufacturing applications (28). The system is composed of a data definition language (DDL), a DDL compiler, and a set of data manipulation primitives (DML) to compile and access either true relational databases or pseudo-relational databases. The system is written in Pascal and allows the use of a large subset of Pascal data types to handle complex objects. An original feature of the system is that it can support a multi-application/multi-database environment. Application programs are written in Pascal. Since the databases are compiled, their data schemata are static.

4.1 Expression of Integrity Constraints

To maintain the consistency or the reliability of any database, some means should be provided so that the integrity constraints could be explicitly specified in the DBMS. This is part of the data definition language.

In this section, we show how integrity constraints can be declared in terms of the four systems mentioned earlier and according to the four-layer classification given in Section 2.1.

4.1.1 Data Typing

The basic level of data integrity is data typing. Each database system provides a limited set of data types which are used to define the nature of attributes.

Let us consider a simple example. Suppose that we want to define the following relation:

PARTS (Partid, Partname, Type, Leadtime, Cost)

where

Partid is a 10-character string
Partname is a 15-character string
Type is a 3-character string
Leadtime is a real number (with two decimals)
Cost is a number representing a dollar amount

Let us see how we can define this relation in terms of the four database systems.

4.1.1.1 Ingres

Ingres offers the following basic data types:

```
C1-C225 for character strings
i1      for 1-byte integers  (-128 to +127)
i2      for 2-byte integers  (-32768 to 32767)
i4      for 4-byte integers  (-2147483648 to 2147483647)
f4      for 4-byte floatings (-10E38 to +10E38)
f8      for 8-byte floatings (-10E38 to +10E38)
date    for dates
money   for dollar amounts
```

These data types are used in Ingres with the command CREATE.

```
CREATE <relationname> (<fieldname> = <datatype>, ...)
```

Therefore, to define the relation PARTS we write:

```
CREATE PARTS (Partid = C10, Partname = C15, Type = C3,
              Leadtime = f4, Cost = money)
```

Successful execution of this CREATE statement causes a new, empty table to be created with the specified name and the specified column definitions. The user can then proceed to enter data into that table using the command APPEND. For example:

```
APPEND PARTS (Partid = 'P1', Pname = 'crank shaft', Type = 'MFG',
              Leadtime = 15.0, Cost = '$100.0')
```

Of course, it is imperative that the given values in the APPEND statement match the types specified in the CREATE statement. If so, the tuple is inserted in the relation PARTS.

4.1.1.2 SQL

SQL offers the following basic data types:

```
NUMBER  for numeric fields (real or integer)
CHAR(n) for n-character strings (n ≤ 255)
DATE    for dates
DOLLAR  for dollar amounts
```

To create the relation Parts with SQL we use the CREATE TABLE command:

```
CREATE TABLE PARTS
(Partid CHAR(10) NONULL,
Partname CHAR(15),
Type CHAR(3),
Leadtime = NUMBER(6,2),
Cost = DOLLAR);
```

where NONULL is used to identify an attribute with no null values.

SQL also offers a SQL INSERT statement to enter tuples of data into the relation.

Remark: Commercial versions of both Ingres and SQL are mainly dedicated to business applications. Therefore, they have a limited set of simple data types. However, there are prototype extensions of those systems developed for engineering applications and supporting the concepts of abstract data types. Since the following two systems have been developed to meet manufacturing requirements, they provide a richer set of more complex data types.

4.1.1.3 RIM

RIM offers the common basic data types used in administration and also supports scalars and arrays of elements commonly used in CAD/CAM. The list of possible data types follows:

DATE	for dates (default is mm/dd/yy but can be changed)
DOLLAR	for dollar amounts (±$99,999,999,999,999.99)
INTEGER	for whole numbers (±999,999,999)
REAL	for real numbers (±1.0E38)
TEXT	for character strings of up to 1500 characters
TIME	for the time in hours, minutes, and seconds
INTEGER n	for an array of n integer numbers
REAL n	for an array of n real numbers

Engineering units of measurement may be declared in RIM. Also supported is a tolerance capability by which users may specify an approximation of equality to be used in qualification operations (WHERE clauses of data language statements).

Declaring our relation example PARTS with RIM requires two steps. First, we must define the attributes of the relations using the ATTRIBUTES

command of the DDL. Second, we can group attributes to form relations using the RELATIONS command. Each attribute is defined by its name, its data type, optionally by a length, and a KEY designation. A relation is defined by its name and its list of attributes. Thus, the example becomes where D> is the prompt of the DDL:

```
D>ATTRIBUTES
D>PARTID TEXT 10 KEY
D>PARTNAME TEXT 15
D>TYPE TEXT 3
D>LEADTIME REAL
D>COST DOLLAR
D>RELATIONS
D>PARTS WITH PARTID PARTNAME TYPE LEADTIME COST
```

4.1.1.4 DBS/R

DBS/R has a powerful data description language (DDL) for describing any true relational data structures as well as sophisticated pseudo-relational data structures, i.e., relations violating the first normal form. The syntax of the DDL is a Pascal-like syntax, and it allows the use of simple data types (INTEGER, REAL, BOOLEAN, CHAR) as well as structured data types (ARRAY, RECORD) and enumerated data types. The complete syntax of this DDL is given in Appendix A.

Data typing is defined by the following rules in Backus-Naur form:

```
<TYPES> ::= "TYPE" <DEFTYPE><SDEFTYPE> ! VOID

<SDEFTYPE> ::= <DEFTYPE><SDEFTYPE> ! VOID

<DEFTYPE> ::= <IDENT> "=" <TYPE> ";"

<TYPE> ::= <IDENT> ! "INTEGER"!"CHAR"!
           "BOOLEAN" ! "REAL" !
           "(" <IDENT><SUITEIDENT> ")" !
           "INTERVAL" ":" <CONST> ".." <CONST>!
           "ARRAY" "[" <TYPE><STYPE>"]""OF"<TYPE>!
           "RECORD" <LISTECHAMPS> "END"

<SUITEIDENT> ::= "," <IDENT><SUITEIDENT> ! VOID

<STYPE> ::= ","<TYPE><STYPE> ! VOID

<LISTECHAMPS> ::= <CHAMPS><CASE>
```

<CHAMPS> ::= <CHAMP><AUTRECHAMP>

<CHAMP> ::= <IDENT><SUITEIDENT> ":" <TYPE> ! VOID

<AUTRECHAMP> ::= ";" <CHAMP><AUTREHAMP> ! VOID

<CASE> ::= "CASE" <IDENTYPE><TYPIDENT> "OF" <LISTECAS> ! VOID

<TYPIDENT> ::= ":" <IDENTYPE> ! VOID

<LISTECAS> ::= <CAS><LCAS>

<CAS> ::= <CONST><LISTECONST> ":" "(" <LISTECHAMPS> ")" ! VOID

<CAS>::= ";" <CAS><LCAS> ! VOID

<LISTECONST> ::= "," <CONST><LISTECONST> ! VOID

<IDENTYPE> ::= <IDENT> ! "INTEGER" ! "CHAR" ! "BOOLEAN"

where

<CONST> represents a constant or an identifier of a constant

<IDENT> represents a valid identifier previously declared

Relations are then declared using a RELATIONS clause, the syntax of which is defined by:

<RELATIONS> ::= "RELATIONS" <RELATION><SRELATION> "END"

<SRELATION> ::= <RELATION><SRELATION> ! VOID

<RELATION> ::= <IDENT> ":" <IDENT> ";"
<CLES><POPULATION><INTEGRITE>

<CLES> ::= "KEYS" ":" <IDENTCHAMP><SIDETCHAMP> ";" ! VOID

<IDENTCHAMP> ::= <IDENT><ACCES>

<SIDENTCHAMP> ::= "," <IDENTCHAMP><SIDENTCHAMP> ! VOID

Using these production rules, we can describe true relational data structures, for example, the relation PARTS,

DB:DATABASE;

```
TYPE
     Tident = ARRAY[INTERVAL:1..10] OF CHAR;
     Tname  = ARRAY[INTERVAL:1..15] OF CHAR;
     Ttype  = ARRAY[INTERVAL:1..3] OF CHAR;
     Tparts = RECORD
                 Partid:Tident;
                 Partname:Tname;
                 Parttype:Ttype;
                 Leadtime:REAL;
                 Cost:REAL;
                 END;
RELATIONS
     Parts:Tparts; KEYS:Partid;
END
```

Moreover, this DDL allows the definition of much more complex data structures involving structured ARRAYs or structured RECORDs and even RECORDs with CASE statements. This is in contradiction with the theory of relational databases since the first normal form can be violated, but it is required by some manufacturing applications to store very complex information (e.g., signals, voice, text, synthetic images).

4.1.2 Referential Integrity

Referential integrity, as discussed in an earlier section, corresponds to the integrity rules which are used to exactly define the set of possible values of some attributes in terms of restriction of their data types. We successively consider this point for each of the four database systems, presenting some generalities.

4.1.2.1 Generalities

In database systems, it is very important to distinguish the difference between a domain on one hand and columns or attributes of a relation on the other. Domain is a collection of values from which the actual values appearing in a given column of a relation are obtained. An attribute, however, represents the use of a domain within a relation. Referential integrity is also called domain integrity and for a given domain is nothing more than the definition of that domain.

A general definition of a domain could be as follows (29):

```
<Domain-Definition> ::= "DCL"<domain-name><key-spec> "DOMAIN"
                        <constraint>
                        <terminator>
```

<key-spec> ::= "PRIMARY" | VOID

<constraint> ::= <data-type><predicate>

<predicate> ::= <predicate-spec> | VOID

<terminator> ::= ";" | "ELSE" <violation-spec>

<violation-spec> ::= <executable-unit>

Looking back at our example and using this syntax we can define the underlying domains of the relation PARTS in a more restricted way. Suppose we impose the first character of any Partid be the letter P, and we only want to store parts for which the leadtime is less than 20 days and the cost is less than $2000. We will write:

```
DCL Partid    PRIMARY DOMAIN CHARACTER(10) VARYING
              SUBSTR(Partid, 1,1) = 'P'
              ELSE
                DO;
                  set return code to
                  "Partid domain rule violated";
                  REJECT
                END;
DCL Partname  DOMAIN CHARACTER (15) VARYING;
DCL Type      DOMAIN CHARACTER (3);
DCL Leadtime  DOMAIN FIXED (5,1)
              Leadtime < 20.0;
DCL Cost      DOMAIN FIXED (5,1)
              Cost > 0.0 AND cost < 2000.0;
```

4.1.2.2 Ingres

Coming back to the CREATE command of Ingres, it becomes obvious that the system Ingres does not support the concept of domain. However, referential integrity option is available in Ingres since an integrity constraint can be defined in the QUEL language for a given column. Therefore, after issuing the CREATE command, a user can use an INTEGRITY statement to specify the range of values for certain attributes of a relation. An integrity constraint may be defined via

"DEFINE""INTEGRITY""ON" Relation-name "IS" qualification

Example:

```
RANGE OF P is Parts
DEFINE INTEGRITY ON P IS P.Leadtime < 20.0
                      AND P.Cost > 0.0 AND P.Cost < 2000.0
```

where P is used instead of Parts (as defined by the RANGE clause).

4.1.2.3 SQL

Similar to Ingres, the CREATE TABLE of the language SQL shows that no distinction is made between attributes and domains. However, SQL also provides similar QUEL features to facilitate the implementation of referential integrity via its ASSERT statement:

"ASSERT"<assertion-name>"ON"<relation-name>":"<predicate>

Example:

```
ASSERT ICI ON PARTS:
Leadtime < 20.0 AND Cost > 0 AND Cost < 2000.0
```

4.1.2.4 RIM

RIM supports the concept of domain in a very primitive way since attributes can be defined independently of the definition of relation and then grouped in various ways to form the relations. However, RIM only offers a limited set of general domains (limited length character-strings, whole numbers, real numbers, dates, dollar amounts, and arrays of real or integer numbers). In addition to the ATTRIBUTES and RELATIONS commands, RIM also provides an optional RULES command for constraining domains. An interesting feature of this command is the possibility of defining a prompt message that will be displayed if the rule is violated at data entry verification time. Therefore, using the following syntax of the RULES command, referential integrity is supported:

```
<RULES> ::= "RULES" <rule-spec>

<rule-spec> ::= <message><attribute-name>
                <predicate><next-rule> ! VOID

<next-rule> ::=          <rule-spec> ! VOID

<message> ::= <prompt-message> ! VOID
```

```
<predicate> ::= <operator><value><next-predicate>
<next-predicate> ::= <connector><predicate> ! VOID
<operator> ::= "EQ" ! "NE" ! "GT" ! "GE" ! "LT" ! "LE" !
               "CONTAINS" ! "EXISTS"

<conector> ::= "AND" ! "OR"
```

Example:

```
D>RULES
D>"Leadtime is greater than 20 days!"  LEADTIME GE 20.0
D>"Cost is less than $2000!"  COST GE 0.0 AND COST LE 2000.0
```

This way, if the attribute COST belongs to several relations, the constraint will apply for each relation.

4.1.2.5 DBS/R

The DDL of DBS/R supports domain integrity as much as Pascal data typing can. A large set of domains can be defined as intervals of values, enumerated sets of values, arrays of elements, OR structured records in addition to the basic domains including Boolean set {True, False}. This has been illustrated in the previous section by the description of the data types. However, due to this syntax and although the concept of internal is supported, it is not possible to define a domain for cost, the values of which would be real numbers ranging from 0.0 to 2000.0. To do that, we can define the attribute cost on the domain of real numbers, and then constrain it to be in the range of 0.0 to 2000.0 using an INTEGRITY clause in the relation definition.

The integrity clause of a <RELATION> rule is:

```
<INTEGRITE> ::= "INTEGRITY" ":" <CHAINE> ";" ! VOID
```

where <CHAINE> is the identifier of a file containing the text of a valid Pascal Boolean function verifying the rule.

Therefore, the relation PARTS becomes:

```
RELATIONS
  Parts:Tparts; KEYS:Partid;
        INTEGRITY:'VERIFPART.PAS';
END
```

where VERIFPART.PAS contains:

```
FUNCTION Verifparts (P:BDT_parts):BOOLEAN;
BEGIN
  Verifparts := ((P.Partid [1] = 'P') AND (P.Leadtime ≤ 20.0)
                 AND (P.Cost > 0.0) AND (P.Cost ≤ 2000.0))
END;
```

Note: In this example we have used the same Boolean function to constrain three attributes of the same relation. Hence, this is really an integrity constraint rather than a domain constraint.

None of the four database systems considered here fully support the concept of referential integrity. Each of these has to make use of some integrity mechanism to restrict the set of possible values for a given attribute, especially if the underlying data type is REAL. However, some may argue that among the three restrictions that we have imposed on the attribution values of Partid, Leadtime, and Cost, two may be viewed as semantic integrity constraints (Leadtime and Cost). If this is true, they can legitimately be declared in an integrity clause as discussed in the next section. Nevertheless, the restriction bearing on Partid is really a domain integrity and it is interesting to notice that the DDL of DBS/R can easily implement it whereas it is not possible with other sytems.

4.1.3 Semantic Integrity

The term semantic integrity in relational databases refers to ensuring that the single attribute, single tuple, intra-relation, and inter-relation integrity checks relevant to a given application field are included in the DBMS. The semantics or meaning of data in a database is a matter of shared understandings among users of the database which should be expressed as integrity constraints according to their understanding of their specific field of concern. In other words, the rules applied to INSERT, UPDATE, DELETE, and MODIFY perform the necessary checking for the above-mentioned class of integrity constraints.

There are four classes of semantic integrity constraints which are as follows:

(1) single attribute
(2) single tuple
(3) intra-relation
(4) inter-relation

These four classes allow us to specify different types of more or less sophisticated semantic integrity constraints.

An application case here, to be introduced in the form of a table, will let us explain better and give an example for each of the classes mentioned before.

Tables representing the operation and accumulation of data related to a machine shop using some CNC and DNC machines come in the following form:

CNC Table

CNC#	TURRET CAP.	# OF AXIS	WORK ZONE, WZ
CNC1	12	3	0.17
CNC2	16	4	0.40
CNC3	8	3	0.04
CNC4	36	6	0.85

WZ = Work Zone, in m^3

CNC 3-Axis

CNC#	X	Y	Z
CNC1	1.25	.45	.30
CNC3	1.00	.35	.10

X, Y, and Z, in m

Given the above tables, we can give an example for each class of semantic integrity constraints:

For simple attribute integrity constraints, which are used to set up the rules for restriction of the domain of an element, we can say that the number of axes of a CNC machine cannot exceed the value of six and it must be a positive number. This is simply because no CNC machines are available with more than six axes.

In single tuple integrity constraints, several attributes of one tuple of a relation are involved. In the above table (CNC 3-axis), work zone for CNC1 with three axes is equal to

$$WZ = X \times Y \times Z$$

where

WZ = Work zone, m^3
X = x axis, m
Y = y axis, m
Z = z axis, m

The third class of semantic integrity constraints are intra-relation integrity contraints which involve attributes from several tuples of a relation. For example, the total work zone to be managed by a cell

controller of a cell which consists of CNC1 and CNC3 is equal to

$$0.30 + 0.25 = 0.55\ m^3$$

Finally, inter-relation integrity constraints involve attributes from various tuples of different relations of the database. Referring to the above example, the total value of tool movement to be used in X axis of CNC1 and CNC3 is (12 × 1.25) + (8 × 1.00) = 23.00.

4.2 Checking of Integrity Constraints

In the previous section, we discussed how to declare integrity constraints in relational databases. In the current section, we are concerned with the problem of verifying those constraints during run-time and, when appropriate, each time a data item is updated or entered into the database.

These kinds of checking methods can be implemented in a DBMS. They are

- immediate checking
- selective checking
- delayed checking

4.2.1 Immediate Checking

Most of the database systems supporting some form of integrity management use immediate checking. This is also true for DBMS's produced by DBS/R.

In immediate checking, each time a tuple is inserted, updated, or deleted, all integrity rules related to elements of this tuple are activated and checked (i.e., evaluated to true or false).

With DBS/R, each time a tuple is entered or modified, a Boolean function LOCALINTEGRITY is invoked in the DBMS code using the value of this tuple, and all integrity rules associated with the relation the tuple belongs to are verified. The integrity rules have been defined in the file indicated in the INTEGRITY clause of this relation (see Section 4.1.2.5). They are expressed in the form of Boolean Pascal functions and they have been included in the Pascal code of the DBMS by the DDL compiler.

SQL, RIM, and Ingres also make use of immediate integrity checking, but the rules are stored in the data dictionary. With Ingres, rules are also associated with one relation, and they are verified each time a tuple of the relation is concerned. With RIM, rules are associated with the

attributes on which they are defined. Therefore, each time a tuple is entered or modified, all rules invoking attributes of this tuple are verified if the CHECK command of RIM is on (see Section 4.2.2.3).

In the case of a general integrity management system of a relational DBMS having a data dictionary and supporting single attribute, single tuple, intra-relation, and inter-relation integrity constraints, the DBMS must keep track in its data dictionary of the relationships between rules, attributes, and relations. This means that rules must be identified by the system (they have a name or a number). Obviously, for tuples involving many rules, integrity checking will slow down the system at execution time, since the DBMS will have to search the appropriate rules in the data dictionary and verify them.

Another way to implement integrity checking can be the use of abstract data types to include the integrity constraints in the records and define the manipulation operations accordingly.

4.2.2 Selective Checking

Some integrity validations may be required only for some database manipulations, i.e., on UPDATE statements only, or on INSERT statements only, on DELETE statements only, or on a combination of them. This is called selective integrity checking.

Selective integrity checking is supported by SQL. For example, we can write

```
ASSERT C18 ON UPDATE of PARTS:
     Leadtime < 20.0 AND Cost > 0 AND Cost < 2000.0
```

Rule C18 will only be verified while updating tuples of relation PARTS.

4.2.3 Delayed Checking

Transaction management in the presence of integrity assertions may require delayed integrity checking since temporary data inconsistencies may be allowed to exist in the database during transaction processing.

Delayed checking was suggested by Lafue (6) and is especially relevant for transactions involving the processing of tuples for which single tuple integrity constraints and intra-relation or inter-relation integrity constraints have to be verified. At some time in the transaction, the single

tuple rule may be valid while the intra- or inter-relation will not be. (For example, in Section 4.1.3 changing the value of X in CNC-3-axis for CNC1 implies the violation of the rule defining the value of WZ in CNC.) Bertino and Apuzzo (19) point out that delaying integrity enforcement to the end of a transaction can be expensive; thus they introduce the notion of integrity enforcement schedule for which integrity constraints can be enforced after the update of a single tuple, after the update of a set of tuples, or at the end of a transaction.

In Section 4.4 we consider other tools under the control of the user or the application program to solve problems associated with transaction management in the presence of integrity constraints.

4.3 Enforcement of Integrity Constraints

Depending on the result of the evaluation of an integrity constraint, the DBMS has to make some decision on what action to take. This is called enforcement of integrity constraints. We can distinguish two types of possible enforcements:

- accept/reject enforcement
- enforcement with ELSE clause

4.3.1 Accept/Reject Enforcement

This is the most simple and straightforward mechanism. If the constraint is evaluated to true, the tuple processing is committed. If the constraint is evaluated to false, the tuple processing is ignored. Ingres, SQL, RIM, and DBS/R use this form of enforcement.

Fenves and Rasdorf (22) propose a mechanism called augmented relation to keep track of the status of the evaluation of integrity rules associated to a relation. This works as follows. If

MEMBER (ID, S, I, d)

is a relation for beams in civil engineering with the following constraint between the beam section S, the moment of intertia I, and the beam depth d

$$S = 2I/d$$

they store the relation

MEMBER (ID, S, I, d, sectionOk)

with

sectionOk := (S = 2 * I/d)

Therefore, each time a new tuple is entered in the database and it verifies the constraint, its field sectionOk is set to true. This implementation allows storage of flagged non-valid tuples which can help for implementing delayed integrity checking, but which can also be of interest for implementing integrity rule enabling and disabling and rule verification enforcement as discussed later in Section 4.4.

4.3.2 Enforcement with ELSE Clause

In his book, Date (1) gives the general syntax of domain integrity rules and of relation integrity rules (see Section 4.1). In both cases, an optional ELSE clause is defined. Therefore, integrity constraint enforcement with an ELSE clause is similar to accept/reject enforcement but some action can be initiated in case of violation of the rule (such as printing of a message on the output device or doing some procedural treatment).

None of Ingres, SQL, or DBS/R use the ELSE clause. RIM can print a message on the computer terminal if a rule is not satisfied when the CHECK command is on. Of course, this message is part of the definition of the rule violated.

4.4 Management of Integrity Constraints

Some of the database systems developed for manufacturing will be used in a very dynamic environment, especially design and manufacturing databases. This means that the database schema may vary over time. This is also true for semantic integrity constraints. Some rules may be valid for any database states, but we can also find situations for which some rules have to be checked at some times and have to be relaxed at other times. Furthermore, in dynamic environments such as engineering environments, engineers may want to add new rules to the system, or delete or update previous or obsolete rules. Finally, the database system should offer the possibility of verifying a given set of rules over a given period in order to maintain or enforce local consistency of the database. Therefore, users and application programs of the database system should have complete control over the management of their integrity checks. To achieve that, the integrity manager of the DBMS should provide features for

- integrity constraints maintenance
- integrity constraints enabling and disabling
- integrity transactions and validation of integrity checkpoints

4.4.1 Integrity Constraints Maintenance

Interactive maintenance of the set of rules must be possible. It will rely on the following commands used whenever the user gets control of the system.

4.4.1.1 ADD Rule

This command is for defining new rules.

Examples: We want to add the following integrity constraint to the existing rule set using the four previous DBMS packages: "The only valid values for Type in PARTS are MFG (manufactured), ASM (assembled), or PUR (purchased)."

Ingres: The DEFINE INTEGRITY command is used:

```
RANGE OF P IS PARTS
DEFINE INTEGRITY ON P IS
P.TYPE = 'MFG' OR P.TYPE = 'ASY' OR P.TYPE = 'PUR'
```

Note: the integrity rule is not identified.

SQL: The ASSERT command is used:

```
ASSERT IC4 ON PARTS:
TYPE IN ('MFG', 'ASY', 'PUR')
```

Note: the integrity rule is identified by its name IC4.

RIM: The RULES command is used:

```
D>RULES
D>TYPE IN PARTS EQ 'MFG' OR TYPE IN PARTS EQ 'ASY' +
+>TYPE IN PARTS EQ 'PUR'
```

Note: the integrity rule is not identified.

DBS/R: not applicable.

4.4.1.2 LIST Rules

This command is for viewing the list of rules currently defined on the database.

Examples:

Ingres: The HELP INTEGRITY command has to be used but it must be followed by one or several relation names.

```
HELP INTEGRITY PARTS
```

will list all the integrity rules defined on relation PARTS and each one is referred to by an integer.

SQL: The definition of each integrity constraint is stored in the SQL data dictionary in, let us say, the relation INTEGRITY. To have the list of existing rules, we must query this relation with a SQL SELECT command.

```
SELECT * FROM INTEGRITY
```

If we are just interested in rules related to the relation PARTS we must add a qualification.

```
SELECT * FROM INTEGRITY
WHERE TNAME = 'PARTS'
```

RIM: Each rule created is stored in a relation called RBSRULES. To view the RULES defined, the SHOW RULES command is used.

```
R > SHOW RULES
```

The list of all integrity rules appear on the terminal and each one is referenced by an integer.

It is also possible to query the relation RBSRULES in a similar way as with SQL.

```
R > SELECT ALL FROM RBSRULES
```

or

```
R > SELECT ALL FROM RBSRULES WHERE NUMRULE LT 10
```

will just list the first 10 rules defined on the database.

DBS/R: not applicable

4.2.1.3 DELETE Rule

This command is for destroying a given rule.

Examples: We want to delete the rule previously added in Section 4.2.1.2.

Ingres: Since integrity constraints are not identified by the user but have an interval integer identifier, we first use the command HELP INTEGRITY tc determine this identifier, then we can use the DESTROY command.

```
HELP INTEGRITY PARTS
assume the rule identifier is 4
DESTROY INTEGRITY PARTS 4
```

If we want to destroy all integrity rules related to PARTS, we enter

```
DESTROY INTEGRITY PARTS ALL
```

SQL: The rules are identified, so it is possible to use the DROP command for the integrity constraints named IC4.

```
DROP ASSERTION IC4
```

RIM: Similar to Ingres, we first list the rules to find out the system identification of the desired rule, and then we can use the DELETE command to remove it from the RBSRULES relation.

```
R>SHOW RULES
assume the rule identifier is 4
R>DELETE ROWS FROM RBSRULES WHERE NUMRULE EQ 4
```

DBS/R: not applicable

4.4.1.4 MODIFY Rules

It should be desirable in some cases to modify the definition of an existing rule. Usually, this command is never implemented because it can always be performed using a DELETE rule command first, followed by an ADD rule command to recreate the constraint with a new definition.

Notice: For efficient implementation of these maintenance commands, it is recommended

- to identify each rule by an identifier
- to store the definition of each rule in the data dictionary

4.4.2 *Integrity Constraints Enabling and Disabling*

The features discussed hereafter have not been implemented in any commercial DBMS so far (1985). They have been experimentally demonstrated at Carnegie-Mellon University in Pascal as an extension of the integrity manager of RIM (23).

There are situations, especially in design environments, for which it is unrealistic to assume that a given integrity constraint must hold in any case, i.e., over all the time. Objects may be designed at varying times and in varying orders. Therefore, during the design process, we know that the object design may be temporarily in an inconsistent state (design not yet completed, partial modification, ...). This is due to the nature of the design process. However, if the computer system maintains integrity checks that are violated in such situations, it will be impossible to complete the design since the computer would reject some of the given values. Therefore, what we need is some mechanism to activate/deactive or turn-on/turn-off a given rule or a specified set of rules, whenever desirable.

4.4.2.1 Enabling Command

If every rule has a unique identifier, the following command can be used by on-line users or application programs to activate one or several integrity constraints:

```
ENABLE <rulename><otherrules>
<otherrules> ::= "," <rulename><otherrules> ! VOID
```

This command sets on the checking mechanism for each subsequent transaction (insertion, deletion, or update) involving tuples of relations on which the integrity constraints apply. By default, we assume that the status of the checking mechanism for a given rule is on.

4.4.2.2 Disabling Command

In a similar way, to turn off the checking mechanism for one or several integrity constraints the following command can be implemented:

```
DISABLE <rulename><otherrules>
```

This command suspends the effect of the specified rule(s), therefore everything works as if the rule(s) was (were) no longer in existence until an ENABLE command is eventually issued. Thus, inconsistent and non-valid data can then be entered into the database under the responsibility of the users of the application programs.

4.4.2.3 The Case of RIM

It can be argued that, if not implemented, the disable capability can always be performed by deleting the rule(s) using the DELETE rule command. Then, if there is the need to reactivate the rule(s), one can always re-create the rule(s) using the ADD rule command. This is the situation with RIM. However, this solution is not always satisfactory because of

- the burden imposed on the user (to recreate the rules)
- the impracticability of this solution in application programs

Therefore, the two previous commands must be implemented in a CAD/CAM database system.

Nevertheless, RIM provides two **global** commands to turn-on/turn-off the integrity checking mechanism:

- CHECK
- NOCHECK

CHECK is a global command which activates all defined constraints. When a new rule is created in RIM, its checking is immediately effective if CHECK is already set to "on." NOCHECK deactivates all defined constraints.

Obviously, this implementation is not suitable in the general situations for which it is desired to have some rules revoked while others are still requested. Unfortunately, many examples of such situations can be found in manufacturing.

4.4.3 Integrity Transactions and Validation of Integrity Checkpoints

The ability of turning-on/turning-off rules at will and the ability to insert inconsistent data into the database raise a new problem: when can the database be assumed to be in a consistent state?

A CAD database will reach a consistent state only at the end of the design process (30, 31). Therefore, we have to try to enforce local integrity only. This can be achieved using several new mechanisms:

- the use of integrity transactions
- rule verification enforcement
- the use of integrity checkpoints

In database terminology, a **transaction** is a sequence of database operations, called an operational unit, which transfers the database from a

consistent state to a new consistent state. Some type of guarantee about the database integrity can be made before or after the transaction processing but not during. Usually, at the end of the transaction, database updates are committed, i.e., validated on the physical database on disk, or rolled-backed, i.e., undone if the logic of the transaction has aborted, i.e., it has finished abnormally or an abort command is found. In conventional database systems, the duration of inconsistency periods is in the order of seconds. In CAD/CAM database systems, it can be in the order of hours or even days (consider for example, finite element analysis programs).

Eastman and Lafue (4) have proposed a refinement of the notion of transaction in the case of local integrity, called integrity transaction. An integrity transaction is a collection of operations on a set of variables (or tuple elements), during which the integrity of these variables, with respect to a set of constraints, is not guaranteed, but before and after which such integrity is either true or not applicable.

In order to reflect the design process nature, integrity transaction can be nested, i.e., a transaction may be composed of a series of non-overlapping transactions. Various nesting and ordering cases of integrity transactions have been considered by Eastman and Lafue (4) in their paper.

Rule verification enforcement can be considered as a special case of integrity transaction. This facility can be used for checking local integrity of the database, i.e., applying one or more constraints to the current state of the database. If integrity rules are uniquely identified (such as in SQL), this facility can be created by implementation of the following command:

INVOKE <rulename><otherrules>

This command causes a batch process to be started for applying the selected set of rules to the set of related tuples of the database. This command assumes that the checking status of the invoked rules is on, otherwise the enabling capability is set automatically, and it also assumes that the tuples are completely defined.

The aim of this command is to locate nonconforming tuples of the database with respect to the rules invoked. Here, three types of implementations are possible.

- With the first solution, all nonconsistent tuples can be located, removed from the database, and recorded in a journal file.

- With the second solution, all nonconsistent tuples can be located flagged in the database (using a status attribute if augmented relations are implemented), and output. This solution has been experimented by Fenves and Rasdorf (23).

- Finally, the third solution is more ambitious since nonconsistent tuples can be located, and numerical values can be given to some ingredient attributes (equality constraints) or bound values can be generated (inequality constraints) for dependent attributes (32).

Global integrity can be checked using the following form of the INVOKE command (it is assumed that no other transactions are running on this database).

```
INVOKE ALL
```

which implies the verification of all rules for the current contents of the database. Depending on the size of the database, the number of rules, and the nature of rules, this process can be quite complex and very time-consuming. However, there is no objection to its implementation except that it can be very complex in the case of cascading verifications. Indeed, there are situations for which the evaluation of some rules may depend on the evaluation of other rules and so on.

Example: Let us consider the three following relations:

```
OBJECT (Oid, Oname, Omass)
PART   (Pid, Volume, Density, Pmass)
COMPOS (Oid, Pid, Qty)
```

and the two following rules expressed in SQL:

```
ASSERT IC1 ON PART:
    Pmass = Volume*Density

ASSERT IC2 ON OBJECT O:
    Omass = (SELECT SUM (Pmass)
             FROM PART
             WHERE Pid IN
                (SELECT Pid
                 FROM COMPOS C
                 WHERE C.Oid = O.Oid))
```

Obviously, if global integrity checking is required, IC1 must be checked before IC2.

Now, to avoid checking of already validated tuples and therefore to speed up the process of global integrity checking, the notion of integrity checkpoint can be introduced.

An **integrity checkpoint** is a special indication recorded on the database log file at the end of a successful INVOKE command. It indicates that since the last database backup or since the last database checkpoint, the specified integrity rules (eventually all rules) are valid up to this integrity checkpoint. Therefore, when integrity checking is enforced for a given rule via an INVOKE command, we only verify tuples modified since the last integrity checkpoint for this rule (cautions have to be taken for rules which depend on other rules). Otherwise we must verify all tuples in the scope of this rule.

An integrity checkpoint is similar but different to a database checkpoint. They are equivalent only just after an INVOKE ALL command.

Integrity checkpoints are only of relevance when the DBMS supports integrity management and there exists some form of integrity constraint enabling and disabling capabilities.

5.0 RECOMMENDATIONS FOR DBS/R

In this last section, we make some recommendations for improving the integrity mechanism of DBS/R, a pseudo-relational database system for manufacturing environments that was developed at the National Research Council of Canada.

5.1 Current Status of DBS/R

DBS/R is a pseudo-relational database system which can be used to create true-relational databases or pseudo-relational databases, i.e., databases incorporating relations which are not in first-normal form and some attributes of which are hierarchically organized.

In fact, DBS/R is composed of three basic components:

- a data description language (DDL) to define new databases
- and a DDL compiler which automatically generates the DBMS code for each database created
- a set of utility programs to establish the communication between the DBMS created and their application programs, so that a multi-user/multi-database environment can be configured.

For each DBMS generated by the DDL compiler, a given set of data manipulation primitives (DML) is created to allow access by application programs (currently written in Pascal).

For a more complete description of the system, see Gagné et al. (28).

The features of the DDL for semantic integrity description have been presented in a previous section. They can be characterized as follows:

- strong data typing is supported (most of Pascal data types are allowed)
- integrity constraints are compiled
- integrity constraints are defined within the relation descriptions with no selective checking capabilities
- single attribute and single tuple integrity constraints can only be described
- integrity constraints are not identified in the system

At the level of the DML, no primitives are provided for integrity constraints management. Integrity rule descriptions are part of the database schema which is static.

At the level of each DBMS, the integrity subsystem has the following features:

- immediate checking is involved on each update
- enforcement of integrity constraints is of yes-or-no type (accepted or rejected)
- no message is returned to the user/application program about the rule checking but a status code is available on request

5.2 Integrity Constraint Expression Reconsidered

In the more general case, an integrity constraint (IC) can be modeled as a 6-tuple as follows (19).

IC = (name,O,T,qual,predicate,F)

where

name	is the identifier of the integrity constraint
O	is the set of data objects on which the rule is enforced
T	is the set of access types, elements of which could be insert, delete, update, or any combination of them
qual	is qualification on elements of O in order to restrict the scope of the rule application
predicate	is the integrity assertion to be verified
F	is a set of actions to be performed by the system if the predicate is evaluated to false

This can be implemented with a SQL-like ASSERT command having the following syntax:

```
<ICassertion> ::= "ASSERT"<rulename>"ON"<Olist>
                  "WHERE"<qualification>
                  "WHEN"<Tlist>
                  "VERIFY"<predicate>
                  "ELSE"<Fexpression>
                  "END" ";"
<Olist> ::= <relationname><otherrelation>
<otherrelation> ::= ","<Olist> | VOID
```

```
<Tlist> ::= "UPDATING" ! "DELETING" ! "INSERTING" ! "ALL" !
            "UPDATING" "," "DELETING" ! "UPDATING" ","
            "INSERTING" ! "DELETING" "," "INSERTING" !
            "INSERTING" "," "DELETING" ! "DELETING" ","
            "UPDATING" ! "INSERTING" "," "UPDATING"
```

Such a command should be implemented in DBS/R as discussed in the following section.

5.3 Proposal for Extensions of DDL in DBS/R

Currently, the definition of an integrity constraint in the DDL of DBS/R can only be done in the description of a relation. We propose to modify the DDL grammar to remove the integrity clause of a relation and to add an optional integrity section at the end of a DDL description allowing the definition of general integrity assertions as defined just before. The DDL syntax can be modified using the following production rules (compare to Appendix A).

```
<AXIOME> ::= <ENTETE><INFORMATION><CONSTS><TYPES>
             <RELATIONS><INTEGRITE> "END"

<RELATIONS> ::= "RELATIONS"<RELATION><SRELATION>

<SRELATION> ::= <RELATION><SRELATION> ! VOID

<RELATION> ::= <IDENT> ":" <IDENT> ";" <CLES><POPULATION>

<INTEGRITE> ::= "INTEGRITY" <RULE><SRULE>

<SRULE> ::= <RULE><SRULE> ! VOID

<RULE> ::= "ASSERT" <IDENT> "ON" <OLIST>
           "WHEN" <TLIST> "VERIFY" <CHAINE>
           <ELSECLAUSE> ";"

<ELSECLAUSE> ::= "ELSE" <CHAINE> ! VOID

<OLIST> ::= <IDENT><SLIST>

<SLIST> ::= ","<OLIST> ! VOID

<TLIST> ::= same as <TLIST> in previous section
```

IDENT represents a valid identifier
CHAINE represents a valid filename

Remark 1: For this implementation, we impose that the assertion predicate is described by a Pascal Boolean function contained in the file the name of which is given by CHAINE following VERIFY. First, this solution has the advantage that it avoids a fairly complicated description of the predicate expression at the DDL syntax level. Second, we can use the full power of the Pascal language to express fairly complicated constraints. Third, since a procedural language is used for expressing the conditional part of the assertion, there is no need for a WHERE clause.

Remark 2: The set of actions of the optional ELSE clause is defined by a Pascal procedure contained in a file, the name of which is given by CHAINE following the reserved word ELSE.

Remark 3: If the DDL descriptions have to be independent of any underlying programming languages, the category <RULE> must be described as <ICassertion> as in the previous paragraph. In this case, a free syntax must be defined for <qualification>, <predicate>, and <Fexpression>. This will notably extend the set of production rules of the DDL syntax, those expressions will have to be translated in terms of the implementation programming language, and finally this will limit the expressive power of the DDL in terms of integrity constraint expression since recursions and controlled iterations will not be possible.

Example:

Using this new DDL syntax, a simplified example of a database description should be:

```
DB:DATABASE;
TYPE

     Tparts = RECORD
                Partid : Tident;
                Partname : Tname;
                Parttype : Ttype;
                Leadtime : REAL;
                Cost : REAL
              END;

RELATIONS

        Parts : Tparts; KEYS : Partid;

INTEGRITY
        ASSERT IC4 ON Parts WHEN ALL
```

```
                    VERIFY 'VERIFLDTM.PAS';
            ASSERT IC5 ON Parts WHEN INSERTING, UPDATING
                    VERIFY 'VERIFCOST.PAS'
                    ELSE 'ERRCOST.PAS'
    END
```

where VERIFLDTM.PAS contains:

```
    FUNCTION Verifleadtime (P : DB_Tparts) : BOOLEAN;
    BEGIN
        Verifleadtime := P.Leadtime<20.0
    END;
```

and VERIFCOST.PAS contains:

```
    FUNCTION Verifcost (P : DB_Tparts) : BOOLEAN;
    BEGIN
        Verifcost := ((P.Cost > 0.0)AND(P.Cost < 2000.0))
    END;
```

and ERRCOST.PAS contains:

```
    PROCEDURE Errcost;
    BEGIN
        WRITELN; WRITELN;
        WRITELN('The cost of part should be in the range of
                $0-2000 !!');
    END;
```

5.4 Proposal for New DML Primitives

In order to provide efficient mangement of integrity constraints, DBS/R should incorporate facilities for rule enabling and rule disabling in its DML, such as

```
ENABLE <rulename><otherrules>
DISABLE <rulename><otherrules>
```

Also, integrity transaction enforcement should be supported through the implementation of a command forcing the batch checking of a specified set of rules. This command could be of the form:

```
INVOKE <rulename><otherrules>
```

For more efficiency on the part of the database management system

processing, the notion of integrity checkpoints can also be implemented as discussed in partid of the section on integrity control.

Furthermore, the Boolean function SUCCESS of the DML can be used to indicate the user/application program the success or failure of a particular primitive related to integrity management (ENABLE, DISABLE, INVOKE). In case of failure (SUCCESS returns false), the primitive REASON could be used to indicate the type of failure (missing rule(s), rule violation, etc.).

5.5 Other Integrity Management Capabilities

It would be nice to provide users and/or application programs with the facilities for maintenance of integrity constraints (such as ADD rules, DELETE rules). This can only be implemented if the DBMS supports a data dictionary management function. Currently, DBMS's created by DBS/R have no need for a data dictionary (since they are compiled systems).

If a completely dynamic integrity management environment is required, a data dictionary can be implemented. However, in this case, this solution brings in the question of current implementation of the integrity manager in the DBMS's. The integrity manager would have to be rewritten for supporting the various commands and updating the data dictionary accordingly. Each DBMS would then become much more flexible and more complete in terms of (semantic) integrity management. On the other hand, the system would become slower due to the overhead of interpreting and translating the user/application commands and then continuously updating and interpreting the data dictionary. Furthermore, a sophisticated syntax has to be defined in the definition of the ASSERT command in order to allow the expression of sophisticated predicates in the integrity constraints.

Therefore, the choice of the implementation strategy of the integrity manager really depends on the application requirements:

- if a fast system with a fixed database schema and complex integrity rule expression is required, the compiled version with no data dictionary is appropriate

- if a fast system with a fixed database schema but with dynamic integrity management is required a hybrid solution consisting of a compiled DBMS with a data dictionary is appropriate

- if a slower system with dynamic database schema and complex integrity rule expression (with dynamic integrity management) is required, an interpreted DBMS with a data dictionary is appropriate. (An example of such a system could be RIM.)

6.0 SUMMARY AND CONCLUSION

This is an extensive report in which several important issues related to integrity constraints have been examined. The literature review provided the basis of work that has been done in this area by other researchers. In addition, some of the proposed techniques, which have been developed and tested before, were introduced.

We also took an in-depth look at the classification of integrity constraints where real world, engineering, and manufacturing worlds as well as database world were examined. The majority of the examples were either directly related to manufacturing environment or came from an engineering field. This classification did let us explain what we really meant for each type of integrity constraints and will obviously lead us to a better solution as well.

The concepts that we discussed under the expression and management of integrity constraints came from an in-depth study of how to manage the problem. Different features of three commercial types of databases, namely INGRES, SQL, and RIM, as well as DBS/R (developed at NRC) were discussed. These discussions led to the development of a conceptual framework and recommendations for implementing an integrity manager in CAD/CAM database systems. These discussions merely concern our prototype system DBS/R. Among those recommendations, we have pointed out that it would be ideal to include the option of maintenance of integrity constraints with such features as ADD, DELETE, ENABLE, and DISABLE rules. This would give us a better chance to invoke the rules required to monitor the integrity of the data.

Furthermore, it was discussed that a useful characteristic of constraint management in design and engineering cases is to delay some checkings until the product is completely designed. However, this may not be appropriate in some cases, such as:

- If the data entered during the process of design and engineering are used for data-dependent values somewhere else and any error in the data entered originally will affect elsewhere, the associated rules must be invoked immediately.

- If the database is used in the manufacturing environment where the data entered may be accessed (and used) anytime during operation, instantaneous validity and accuracy of data are important issues.

Finally, this report has prepared the groundwork and most of the information required for the future development and implementation of an

integrity manager in a relational database. It should be kept in mind that the process of development, testing, and final evaluation of integrity assertion systems also needs an indepth knowledge of the problem and careful attention at two levels:

(1) at the physical level, since adequate mechanisms must exist in the DBMS in order to efficiently describe and manage the integrity rules

(2) at the logical level, since appropriate and efficient rules must be defined and correctly invoked according to the logic of the application field under concern

REFERENCES

1. C.J. Date. An Introduction to Database Systems. 3rd ed. Addison-Wesley, Reading, MA. 1982.

2. W.A. Kent. A simple guide to fine normal forms n relational database theory. ACM Commun. 26(2): 120-125; 1983.

3. H.C. Smith. Database design: composing fully normalized tables from rigorous dependency diagram. ACM Commun. 28(8): 826-838; 1985.

4. C.M. Eastman and G.M.E. Lafue. Semantic integrity transactions in design databases. In File Structures and Data Bases for CAD. North-Holland, Amsterdam. 1981. pp. 45-57.

5. Boeing Computer Services Co. BCS Rim Relational Information Management System version 6.0 Users Guide. Boeing Computer Services Co., P.O. Box 24346, Seattle, WA. 1983.

6. G.M.E. Lafue. An approach to automatic maintenance of semantic integrity in large design databases. Proceedings of the National Computer Conference, AFIPS-48, Montreal, P.Q. AFIPS Press, 1979. pp. 713-716.

7. R.A. Frost et al. A step towards the automatic maintenance of the semantic integrity of databases. Comput. J. 26(2): 124-133; 1983.

8. G.M.E. Lafue. Semantic integrity dependencies and delayed integrity checking. Proceedings of the 8th International Conference of Very Large Databases, Mexico City, Mexico. September 1982. pp. 292-294.

9. L. Cholvy et al. A CAD object-oriented and rule-based systems. Proceedings of Information Processing Conference, France. 1983. pp. 501-505.

10. P.A. Bernstein et al. Fast maintenance of semantic integrity assertions using redundant aggregate data. Proceedings of the 6th International Conference on Very Large Databases, Montreal, P.Q. 1980. pp. 126-136.

11. C.M. Eastman and K. Preiss. A review of solid shape modeling base on integrity verifications. J. Comput.-Aided Des. 16(2): 66-80; 1984.

12. J.M. Nicolas. Logic for improving integrity checking in relational data bases. Acta Inf. 18: 227-253; 1982.

13. W. Weber et al. Integrity checking in database systems. J. Inf. Syst. 8(2): 125-136; 1983.

14. A.B. Cremers et al. AIM, an integrity monitor for the data base system Ingres. Proceedings of the 9th International Conference on Very Large Databases, 1983. pp. 167-170.

15. M. Stonebraker. Implementation of integrity constraints and views by query modification. ACM SIGMOD, San José, CA. 1975. pp. 65-78.

16. D.Z. Badal. Semantic integrity enforcement in centralized DBMS and distributed DBMs based on slow and post networks. J. Inf. Syst. 9(2): 121-138; 1984.

17. N. Parkash et al. Specifiying integrity constraints in a network DBMS. Comput. J. 27(3): 209-217; 1984.

18. T.W. Ling et al. A method to eliminate avoidable checking of integrity constraints. IEEE Proceedings of Trends and Application. 1984.

19. E. Bertino and D. Apuzzo. Integrity aspects in database management systems. IEEE Proceedings of Trends and Applications. 1984. pp. 43-52.

20. E. Simon and P. Valduriez. Efficient algorithms for integrity control in a database machine. IEEE Proceedigns of Trends and Applications. 1984. pp. 53-59.

21. L. Lilien and B. Bhargava. A scheme for batch verification of integrity assertions in a database system. IEEE Trans. Software Eng. SE-10(6): pp. 664-680; 1984.

22. S.J. Fenves and W.J. Rasdorf. Role of database management systems in structural engineering. Tech. Report, DRC-12-13-82. Design Research Center, Carnegie-Mellon University, Pittsburgh, PA. December 1982.

23. S.J. Fenves and W.J. Rasdorf. Treatment of engineering design constraints in a relational data base. Eng. Comput. 2: 27-37; 1985.

24. S.J. Fenves. Representation and processing of engineering design constraints in a relational database. COMPINT '85, Montreal, P.Q. 1985.

25. M.M. Astrahan et al. System R: relational approach to database management. ACM Trans. Database Syst. DS-1(2): 97-137; 1976.

26. D.D. Chamberlin et al. SEQUEL2: a unified approach to data definition, manipulation, and control. IBM J. Res. Dev. 20(6): 560-575; 1976.

27. M.M. Astrahan et al. A history and evaluation of system R. IBM Res. Rep. RJ843 (36129), IBM Res. Lab., San José, CA. 1980.

28. M. Gagné, F. Vernadat, and P.W.U. Graefe. Design and implementation of a relational database for a manufacturing cell. CICS'84 Conf., Canadian Industrial Computer Society. 1984.

29. Date, C.J. An introduction to database systems. Vol. II. Addison-Wesley, Reading, MA. 1984.

30. C.M. Eastman and A.R. Kutay. Transactions and concurrency in engineering databases. Tech. Report, Designs Research Center, Carnegie-Mellon University, Pittsburg, PA. 1982.

31. T. Neumann and C. Hornung. Consistency and transactions in CAD database. Proceedings of 8th Very Large Databases. International Conference Mexico City, Mexico, September 8-10, 1982. pp. 181-188.

Additional Reading List

1. R.P. Braegger et al. Gambit: An internative database design tool for data structure, integrity constraints and transactions. IEEE Trans. Software Eng. SE-11(7): 574-582; 1985.

2. D.A. Burne. A numberless, tensed language for action oriented tasks. Carnegie-Mellon University, Pittsburg, PA. 1982. CMU-RI-TR-82-12.

3. C.G.N. Dampney. Precedency control and other semantic integrity issues in a workbench database. Database Week, Engineering Design Applications. 1983.

4. E.B. Fernandez et al. Databases security and integrity. Addison-Wesley Reading, MA. 1981.

5. H. Gallaire et al. An overview and introduction to logic and databases. Logic and Databases, Plenum Press, New York, NY. 1984.

6. R. Greer. An introduction to sampling to estimate database integrity. Proceedings of the 2nd International Workshop on Statistical Database Management. September 1983. pp. 36-67.

7. G.D. Held, M.R. Stonebraker, and E. Wong. Ingres: a relational data base system. Proc. Nat. Comput. Conf., AFIPS-44. AFPS Press, Montvale, NJ. 1975. pp. 409-416.

8. J. Mylopoulos, P.A. Bernstein, and H.K.T. Wong. A language facility for designing database-intensive applications. ACM Trans. Database Syst. 5: 287-296; 1980.

9. J.M. Nicolas and K. Yazdonion. Integrity checking in deductive data bases. Logic and Databases, Plenum Press, New York. 1978.

10. N. Roussopoulos et al. CSDL: A conceptual schema definition language for the design of database applications. IEEE Trans. Software Eng. SE-5: 481-496; 1975.

11. M. Stonebraker, E. Wong, and P. Kreps. The design and implementation of Ngres. ACM Trans. Database Syst. 1(3): 189-222; 1976.

12. G. Wiederhold. Database Design. 2nd ed. McGraw-Hill Book Co., New York, NY. 1983.

APPENDIX A

Complete syntax of the data definition language (DDL) of DBS/R.

```
1   SYNTAXE
2
3
4   <AXIOME> = <ENTETE><INFORMATIO><CONSTS><TYPES><RELATIONS>
5
6           $
7
8   <MOTCLES> = ":" ! ";" ! "-" ! "=" ! "+" ! "(" ! ")" ! ".."
9
10              ! "[" ! "]" ! ","
11
12              ! "ACCESS"
13
14              ! "ARRAY" ! "BOOLEAN" ! "CHAR" ! "CASE"
15
16              ! "CLASS" ! "CONST"
17
18              ! "DATABASE" ! "DBA" ! "END" ! "ESTIMATED"
19
20              ! "FALSE" ! "FAST"
21
22              ! "INTEGER ! "INTEGRITY" ! "INTERVAL" ! "KEYS"
23
24              ! "NAVIGATORS"
25
26              ! "OF" ! PASSWORD" ! "PER" ! "POPULATION" ! "REAL"
27
28              ! "RECORD" ! "RELATIONS" ! "TRUE" ! "TYPE"
29
30              ! "USERNAME" ! "VERY"
31
32  <ENTETE> = <IDENT> ":" "DATABASE" ";"
33
34          $
35
36  <INFORMATIO> = "DBA" ":" "PASSWORD" "=" <CHAINE> ";"
37
38                 "USERNAME" "=" <CHAINE> ";"
39
40                 "NAVIGATORS" ":" <ENTIER> ";"
41
42          $
```

```
58

 43
 44     <CONSTS> = "CONST" <CONSTDEF><SCONSTDEF> ! VIDE
 45
 46               $
 47
 48     <SCONSTDEF> = <CONSTDEF><SCONSTDEF> ! VIDE
 49
 50               $
 51
 52     <CONSTDEF> = <IDENT> "=" <CONST> ";"
 53
 54               $
 55
 56     <CONST> = <SIGNE><NOMBRE> ! <CHAINE> ! "FALSE" ! "TRUE"
 57
 58               $
 59
 60     <SIGNE> = "+" ! "-" ! VIDE
 61
 62               $
 63
 64     <NOMBRE> = <ENTIER> ! <IDENT>
 65
 66               $
 67
 68     <TYPES> = "TYPE" <DEFTYPE><SDEFTYPE> ! VIDE
 69
 70               $
 71
 72     <SDEFTYPE> = <DEFTYPE><SDEFTYPE> ! VIDE
 73
 74               $
 75
 76     <DEFTYPE> = <IDENT> "=" <TYPE> ";"
 77
 78               $
 79
 80     <TYPE> = <IDENT> ! "INTEGER" ! "CHAR" ! "BOOLEAN"
 81
 82              ! "REAL"! "(" <IDENT><SUITEIDENT> ")"
 83
 84              ! "INTERVAL" ";"<CONST> ".." <CONST>
 85
 86              ! "ARRAY" "[" <TYPE><STYPE> "]" "OF" <TYPE>
 87
 88              ! "RECORD" <LISTECHAMPS> "END"
```

```
89
90              $
91
92      <SUITEIDENT> = "," <IDENT><SUITEIDENT> ! VIDE
93
94              $
95
96      <STYPE> = ","<TYPE><STYPE> ! VIDE
97
98              $
99
100     <LISTECHAMPS> = <CHAMPS><CASE>
101
102             $
103
104     <CHAMPS> = <CHAMP><AUTRECHAMP>
105
106             $
107
108     <CHAMP> = <IDENT><SUITEIDENT> ":" <TYPE> ! VIDE
109
110             $
111
112     <AUTRECHAMP> = ";" <CHAMP><AUTRECHAMP> ! VIDE
113
114             $
115
116     <CASE> = "CASE" <IDENTYPE><TYPIDENT> "OF" <LISTECAS> ! VIDE
117
118             $
119
120     <TYPIDENT> = ":" <IDENTYPE> ! VIDE
121
122             $
123
124     <LISTECAS> = <CAS><LCAS>
125
126             $
127
128     <CAS> = <CONST><LISTECONST> ":" "(" <LISTECHAMPS> ")" ! VIDE
129
130             $
131
132     <LCAS> = ";" <CAS><LCAS> ! VIDE
133
134             $
```

```
135
136     <LISTECONST> = "," <CONST><LISTECONST> ! VIDE
137
138             $
139
140     <RELATIONS> = "RELATIONS" <RELATION><SRELATION> "END"
141
142             $
143
144     <SRELATION> = <RELATION><SRELATION> ! VIDE
145
146             $
147
148     <RELATION> = <IDENT> ":" <IDENT> ";"<CLES><POPULATION><INTEGRITE>
149
150             $
151
152     <CLES> = "KEYS" ":" <IDENTCHAMP><SIDENTCHAMP> ";" ! VIDE
153
154             $
155
156     <POPULATION> = "ESTIMATED" "POPULATION" ":" <ENTIER> ";" ! VIDE
157
158             $
159
160     <INTEGRITE> = "INTEGRITY" ":" <CHAINE> ";" ! VIDE
161
162             $
163
164     <IDENTCHAMP> = <IDENT><ACCES>
165
166             $
167
168     <SIDENTCHAMP> = "," <IDENTCHAMP><SIDENTCHAMP> ! VIDE
169
170             $
171
172     <IDENTYPE> = <IDENT> ! "INTEGER" ! "CHAR" ! "BOOLEAN"
173
174             $
175
176     <ACCES> = "[" "ACCESS" ":" <VITESSE> "]" ! VIDE
177
178             $
179
180     <VITESSE> = "FAST" ! "VERY" "FAST" ! <ENTIER> "PER" "CLASS"
181
182             $
```